JavaScript: ¿Cuándo debo usar forEach y map?

Mario Souto
02/06/2021

Siempre que avanzo en las clases de los cursos de JavaScript, noto que en el momento en que dejo los loop convencionales y paso a un forEach, varios alumnos se pierden un poco al principio y la idea de esta publicación es eternizar esta explicación en texto, ayudando tanto a los que son mis alumnos o alumnas, como a ti que puedes estar con esta duda ahora.

Introducción

Si toma los 3 códigos a continuación, todos tendrán el mismo retorno en el navegador.

```
const nombres = ['Whinds', 'Freeway', 'Test', 'Maria'];

for(let i = 0; i < nombres.length; i = i + 1 ) {
    console.log('[for]', nombres[i]);
}

const nombres = ['Whinds', 'Freeway', 'Test', 'Maria'];

nombres.forEach(function(nombre, i) {
    console.log('[forEach]', nombre, i);
})
```

```
const nombres = ['Whinds', 'Freeway', 'Test', 'Maria'];

nombres.map(function(nombre, i) {
    console.log('[forEach]', nombre, i);
})
```

Al principio, puedes asumir que si el resultado es el mismo, hacen lo mismo. Pero está claro si miramos un poco el `forEach` y el `map` que aquí no tenemos que preocuparnos con una variable de índice (que solemos llamar i), que sirve para controlar el looping ya que es un elemento que normalmente se incrementa o disminuye. Sin embargo, es fácil notar que, si bien no lo escribimos, este dato es accesible para nosotros solo esta vez lo recibimos como *argumento de la función que pasamos al `forEach` y al `map`*. Para aclarar esto, echemos un vistazo a una posible implementación de forEach.

La implementación no funciona en el modelo array.map y *array.forEach*, *para eso tendríamos que lidiar con [prototype](#) y solo eso ya rinde una publicación.*

Entendiendo forEach

Anteriormente vimos que forEach pasa por todos los elementos de un array, al igual que el loop for normal, esto se debe a que internamente tiene un lop for. Si fuéramos a implementarlo, se vería como el siguiente código:

```
function nossoForEach(arr, funcion) {
    for(let i = 0; i < arr.length; i = i + 1) {
        funcion(arr[i], i);
    }
}

nuestroForEach(['nombre', 'nombre2'], function(nombre, indice) {
    console.log(nombre, indice)
})
```

Siempre que vaya a hacer un loop for, vale más la pena usar un forEach, ya que elimina la carga mental de tener que lidiar con variables de control y, por lo tanto, puede ayudar a

que el código sea más fácil de leer, teniendo en cuenta que es una forma muy usada en el mundo JavaScript en general.

Entendiendo map

En la práctica, como vimos al principio de la publicación, el map y el forEach parecen hacer lo mismo, pero la diferencia viene cuando analizamos el *retorno* por lo que sale de ellos, la diferencia es clara.

```
const nombres = ['Whinds', 'Freeway', 'Test', 'Maria'];

const retornoForEach = nombres.forEach((nombreAtual) => {
    console.log(nombreActual);

  return nombreActual.toUpperCase();
})
console.log(retornoForEach) *// undefined*

const nombres = ['Whinds', 'Freeway', 'Test', 'Maria'];

const retornoMap = nombres.map((nombreActual) => {
    console.log(nombreActual);

  return nombreActual.toUpperCase();
})
console.log(retornoMap) *// ['WHINDS', 'FREEWAY', 'TEST', 'MARIA']*
```

Si bien forEach se creó para ser una alternativa al loop for, el map se creó para realizar una operación de transformación / cambio en los elementos de una array y al final de estas operaciones para tener una nueva lista con la misma cantidad de elementos que la lista base.

Si tuviéramos que implementar el map, caeríamos en un código en esta línea:

```
function nuestroMap(arr, funcion) {
    const nuevoArray = [];
    for(let i = 0; i < arr.length; i = i + 1) {
```

```
            nuevoArray.push(funcion(arr[i], i));
        }    return nuevoArray
    }

    nuestroMap(['nombre', 'nombre2'], function(nombre, indice) {
        console.log(nombre, indice)
    })
```

Esto nos facilita concatenar operaciones como hacer un map y unir con un filter, ya que el retorno del map es un array, podemos escribir un código en esta línea: `.map() .filter()`.

Conclusión

Eso es todo, espero que haya quedado claro que si solo queremos una forma más elegante de trabajar con for, usamos el `.forEach` y si queremos transformar/cambiar valores o incluso concatenar operaciones en arrays el `.map` es el más adecuado.

Si te interesa este tipo de contenido tienes que conferir los cursos de JavaScript en la plataforma de Alura Latam.

ARTÍCULOS DE TECNOLOGÍA > FRONT END